MANUAL DE USUARIO

APLICACIÓN BASE PARA EL DESARROLLO EN J2EE

Edición: **V 1.4**

Autor: **Unidad de Informática Corporativa**
Área de Desarrollo y Mantenimiento

Fecha: Mayo 2006

Historia del Documento			
Versión: **Beta**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta.		
	Elaborado por:	**Marta Martín Jiménez**	*Fecha*: **Diciembre 2003**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Diciembre 2003**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Diciembre 2003**
Versión: **1.0**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta.		
	Elaborado por:	**Marta Martín Jiménez**	*Fecha*: **Febrero 2004**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Marzo 2004**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2004**
Versión: **1.2**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas (con nuevos anexos).		
	Elaborado por:	**José Manuel González Martín**	*Fecha*: **Enero 2005**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Enero 2005**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Enero 2005**
Versión: **1.3**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Diego García Carrera**	*Fecha*: **Febrero 2005**
	Revisado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2005**
			Fecha:
			Fecha:
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2005**
Versión: **1.4**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Diego García Carrera**	*Fecha*: **Marzo 2006**
	Revisado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2006**
			Fecha:
			Fecha:
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2006**

Lista de distribución del documento

Nombre	Localización

Índice

1 INTRODUCCIÓN 5
1.1 Objetivo del documento 5
2 ESTRUCTURA DE LA PROPUESTA 6
2.1 Estructura de la aplicación base 6
3 GENERAR APLICACIÓN BASE 8
3.1 Consideraciones Previas 8
3.2 Ejecución de la herramienta 8
3.2.1 Creación del esqueleto base de la aplicación 8
3.2.2 Generación de Objetos Valor y Objetos de acceso a datos 10
3.2.3 NOVEDAD: Extensión Modulo GIS 12
4 PERSONALIZAR LA APLICACIÓN BASE 13
4.1 Sistema de Logs. Cambios en WEB-INF\config\log.xml 13
4.2 Propiedades de la aplicación. Cambios en WEB-INF\config\ app-config.properties 14
4.2.1 Propiedades generales de la aplicación 14
4.2.2 Propiedades relativas al sistema de seguridad 14
4.2.3 Otras propiedades 15
4.2.4 Añadir propiedades a nuestra aplicación 15
4.3 Descriptor de controlador Struts. Archivo WEB-INF\struts-config.xml 15
4.4 Otras consideraciones 16
5 CARACTERÍSTICAS DE LA APLICACIÓN GENERADA 16
6 DESCRIPCIÓN DE LOS COMPONENTES PRINCIPALES DISPONIBLES 19
6.1 Personalización en la secuencia de arranque 19
6.1.1 Ejecución de código al inicio o fin de la aplicación. Clase XXXConfigApp 19
6.2 Planificador de tareas 20
6.3 Sistema de Log Estándar (log4j) 23
6.3.1 Uso del sistema de Log en la aplicación base 23
6.4 Validaciones basadas en Struts. 25
6.4.1 Pasos para aplicar la validación basada en Struts 25
6.4.2 Actuaciones en los XML con las reglas de validación 25
6.4.3 Actuaciones en los ActionsForms y AplicacionResources.properties 26
6.4.4 Actuaciones en la JSP del formulario y validaciones en cliente 27
6.5 Gestión de transacciones en aplicación Base. 29
6.6 NOVEDAD. Gestión dinámica de configuración de la aplicación 31
6.6.1 Introducción 31
6.6.2 Guía Rápida de uso 31

7 ANEXO I 34
7.1 EJEMPLO 1. Pasos para utilizar PreparedStatement en los OAD's 34

1 INTRODUCCIÓN

1.1 Objetivo del documento

La Aplicación Base para el desarrollo de aplicaciones Web en J2EE se trata de una esqueleto de aplicación que incorpora por defecto funcionalidades básicas y el comportamiento estándar que deben seguir las aplicaciones J2EE elaboradas según los estándares de la JCyL.

El objetivo que se pretende alcanzar con este desarrollo es que las aplicaciones de gestión se puedan construir a partir de esta aplicación base sin más que ampliar la funcionalidad específica. De esta manera todas las aplicaciones se construirán sobre la misma arquitectura lo que garantizará que ciertos comportamientos sean homogéneos para todas las aplicaciones.

Este proyecto posee un conjunto de características que son la aplicación de los estándares de desarrollo Web corporativo en la Junta de Castilla y Leon, así como un amplio número de recursos y herramientas que permiten mejorar la productividad a la hora de afrontar un desarrollo Web a medida y de calidad.

Las aplicaciones generadas a partir de este esqueleto base están asociadas con un completo sistema de seguridad que autorizan la interacción únicamente a los usuarios especificados y a las acciones previamente descritas. Además asegura que cumplen con los estándares corporativos de desarrollo Web se beneficiarán de disponer de:

- Una gestión eficiente de los ficheros de configuración.
- Un Escritura en los ficheros de log.
- Una Organización del proyecto en directorios y ficheros
- Una calidad en la organización y estilos de la capa de presentación.
- Una arquitectura interna escalable y fácilmente de mantener.

2 ESTRUCTURA DE LA PROPUESTA

2.1 Estructura de la aplicación base

La propuesta contiene la estructura propia de una aplicación Web generada con los estándares de la JCYL:

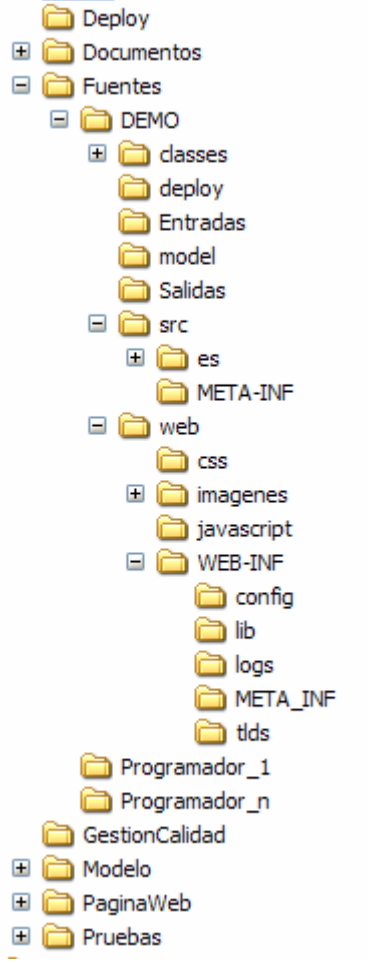


Figura 1: Estructura Aplicación Base

- Componentes de presentación:
  - hojas de estilo en el directorio **css**
  - imágenes en el directorio **imagenes**
  - scripts de procesamiento en cliente en el directorio **javascript**
  - páginas HTML y páginas JSP a partir del directorio base de la aplicación
- Librerías necesarias, en el directorio **lib.** Se incluyen las librerías de struts versión 1.1, librería de log4j, y librerías propias de la aplicación base: **jcylfw-X.X.jar**, **jcylutiles-X.X.jar, jcylbd-1.0.jar** que contiene las clases propias del framework (en su versión correspondiente) y **SistSegu-X.X.X.jar** librería del sistema de seguridad (usando la versión correspondiente).
- Clases java de la aplicación base, en el directorio **src**

- Archivos de configuración:

- **WEB-INF\web.xml**: para definir la aplicación Web según el estándar DTD Web Application 2.3 definido en http://java.sun.com/dtd/web-app_2_3.dtd

- **WEB-INF\config\ConfigurationRepository.xml:** que contiene la referencia a todos los ficheros de configuración de la aplicación, al que llamamos repositorio de configuración. Mas detalles en el apartado correspondiente

- **WEB-INF\nombreAplicacion-config.xml**: donde se definen las características necesarias para el correcto funcionamiento del framework **struts** según el estándar DTD Struts Configuration 1.1 definido en http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd
- **WEB-INF\config\log.xml**: que especifica la configuración del sistema de Log, basado en el componente **log4j**.
- **WEB-INF\config\log4jfw.properties:** que especifica la configuración del log interno al sistema.
- **WEB-INF\config\app-config.properties**: donde se puede parametrizar la aplicación a desarrollar, mediante la especificación de parámetros en forma nombre-valor en un archivo **properties** propio de java.
- **WEB-INF\config\JCYLPlanificador.properties**: que especifica la configuración de las tareas que se deseen planificar en el tiempo y sus características.
- **WEB-INF\tiles-def.xml** : donde se definen la organización de las páginas jsp en el caso de utilizar Tiles Struts (opcional y no utilizado de momento)
- **WEB-INF\validation.xml** : donde definir las reglas de validación (usado en el formulario de Login)
- **WEB-INF\validator-rules.xml** : definición de las reglas de validación en cliente basadas en javascript.
- **WEB-INF\config\ConfigurationRepository.xml:** que contiene la referencia a todos los ficheros de configuración de la aplicación, al que llamamos repositorio de configuración

3 GENERAR APLICACIÓN BASE

En este apartado describiremos paso por paso como comenzar a desarrollar una nueva aplicación J2EE a partir de esta aplicación base.

3.1 Consideraciones Previas

Para poder ejecutar el resultado generado por la herramienta descrita posteriormente se debe tener en cuenta que tal producto se basa o utiliza **el sistema de seguridad para aplicaciones J2EE**. Este sistema se basa en un repositorio de datos independiente donde alojar los usuarios, permisos y descripción de la aplicación así como unos componentes de acceso al mismo.

Por lo tanto en este sistema de seguridad estar dados de alta al menos los elementos básicos de los que se compone la aplicación que son:

1. Registro con el nombre de la aplicación
2. Datos de un Usuario con acceso a esta aplicación con un Rol de acceso.
3. Al menos una función que pertenezca a esta aplicación y accesible por el usuario de entrada al sistema

El almacenamiento de estos objetos puede estar basado en **Base de Datos** o en **ficheros XML.** El Sistema de Seguridad principal es el basado en **Base de Datos**. La versión en ficheros **XML** está disponible para facilitar el desarrollo sin tener instalada la infraestructura completa de Seguridad.

Para conocer en que consiste este sistema tanto versión BD como XML consultar el **Manual de Usuario del Sistema de Seguridad.**

3.2 Ejecución de la herramienta

3.2.1 Creación del esqueleto base de la aplicación

Esta aplicación se debe ejecutar desde la línea de comandos, debemos tener instalado JDK 1.3 o superior.

Consta de los siguientes ficheros:

GenerarAplicacionJ2EE1_4.jar

GenerarAplicacionJ2EE.properties

GestSeguXMLv1.exe

NombreAplicacion.zip

SeguridadBD.zip

SeguridadXML.zip

VariablesGenerarWin.properties

WebApp.zip

WebGIS.zip

Para ejecutar el programa deberá teclear en línea de comandos lo siguiente: (siendo **X_X : Versión actual)**

```
java –jar GenerarAplicacionJ2EEx_x.jar
```

Nos mostrará la siguiente pantalla y nos pedirá una serie de parámetros:

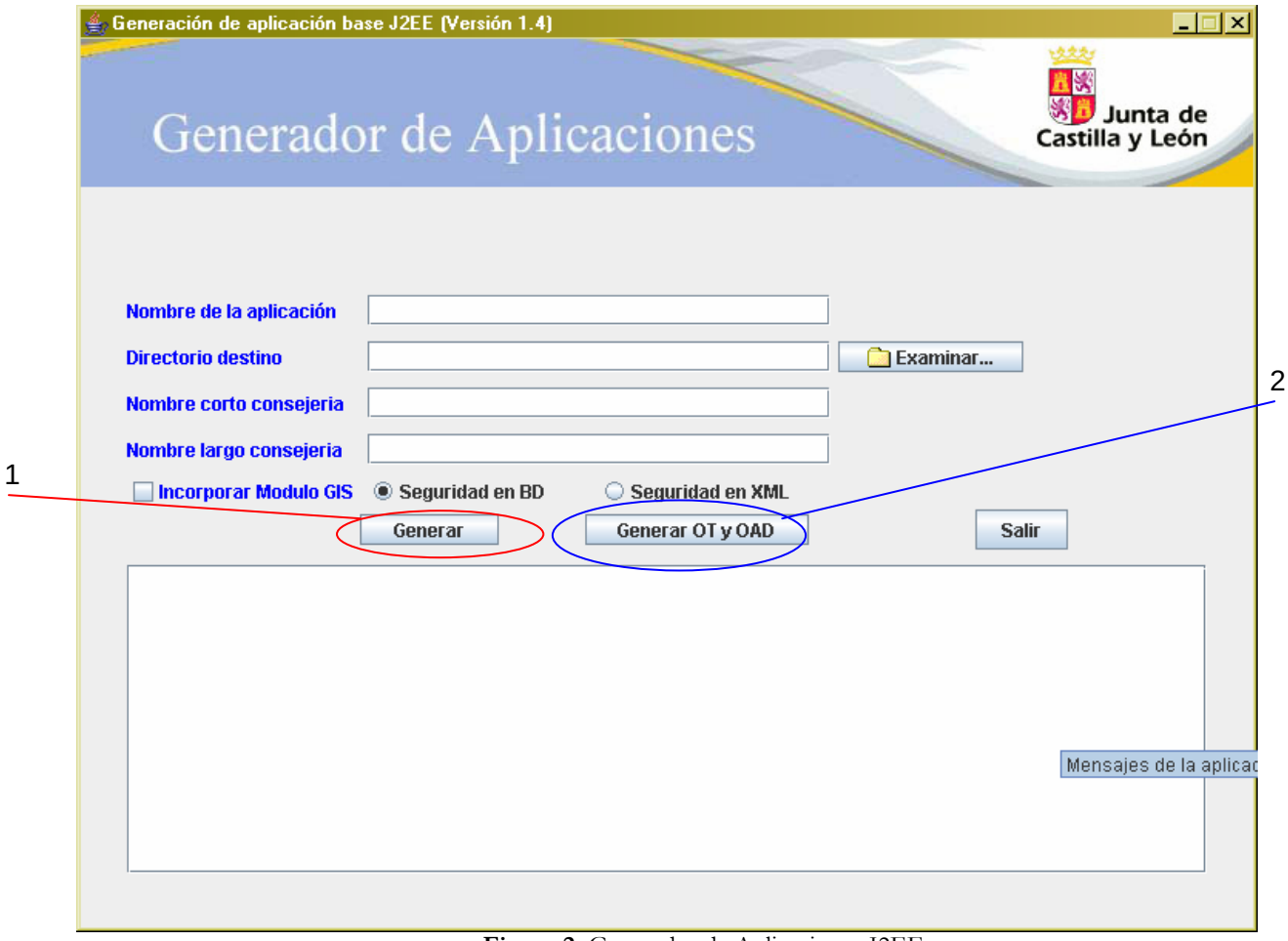


Figura 2: Generador de Aplicaciones J2EE

Parámetros necesarios (todos los campos son obligatorios):

- **Nombre de la aplicación:** *Nombre corto de 4 letras (Mayúsculas), utilizado como identificador de la aplicación que debe coincidir con el introducido en el sistema de seguridad.*
- **Directorio destino:** *Directorio donde queremos generar la aplicación. Deberá estar vacío.*
- **Nombre corto consejería:** *Iniciales de la consejería (minúsculas), necesario para el nombrado de los paquetes java del código fuente generado*. *Ej: cf, gss, ce, etc.*
- **Nombre largo consejería :** *Nombre completo, descriptivo para mostrarlo en las páginas de la aplicación*
- **Seguridad en BD :** *Incluir las librerías de acceso y ficheros necesarios del sistema de seguridad en la versión Base de Datos.*
- **Seguridad en XML :** *Incluir las librerías de acceso y ficheros necesarios del sistema de seguridad en la versión XML*
- **Incorporar Modulo GIS:** Casilla opcional para añadir al esqueleto el módulo GIS

A continuación debemos pulsar el botón ***Generar***[1] y cuando el proceso haya acabado nos mostrará una ventana indicándolo.

3.2.2 Generación de Objetos Valor y Objetos de acceso a datos

Opcionalmente, pero siempre después de tener generada la aplicación, podemos ejecutar la generación de OT y OAD desde el botón GenerarOTyOAD o mediante la siguiente instrucción en línea de comandos:

```
java –jar GenerarAplicacionJ2EEX_X.jar
```

Pulsando el botón **Generar OT y OAD**[2]

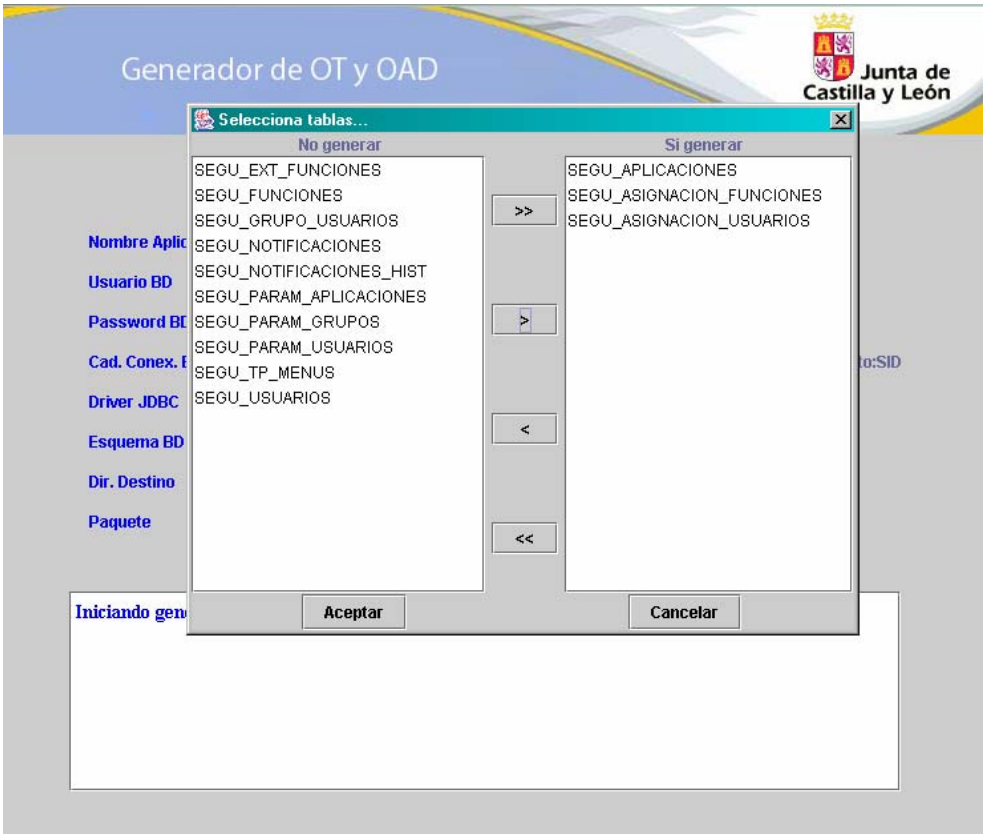
Ahora nos mostrará la siguiente pantalla:

Figura 3: Generador de OT y OAD

Los datos que debemos introducir son:

- **Nombre de la aplicación:** *Nombre corto de 4 letras (Mayúsculas), utilizado como identificador de la aplicación que debe coincidir con el introducido en el sistema de seguridad. Además debe coincidir con lo introducido en el apartado anterior.*
- **Usuario BD :** *Código de Usuario de acceso a los datos de las tablas.*
- **Password BD:** Clave de acceso.
- **Cad. Conex. BD:** *Cadena JDBC para especificar las caracteristicas de conexión.*
- **Driver JDBC :** *Driver de acceso, por defecto, driver de Oracle.*
- **Esquema BD :** *Esquema de base de datos (Mayúsculas) donde residen las tablas.*
- **Dir. destino:** *Se refiere al **directorio src**. Debemos indicar donde se encuentra.*
- **Paquete:** *Debemos indicarlo de la siguiente forma:*
  es.jcyl.nombreconsejeria.nombreaplicacion

Una vez introducidos los parámetro de conexión nos mostrará las tablas del esquema para poder elegir sobre cual generar las clases OAD y OT.

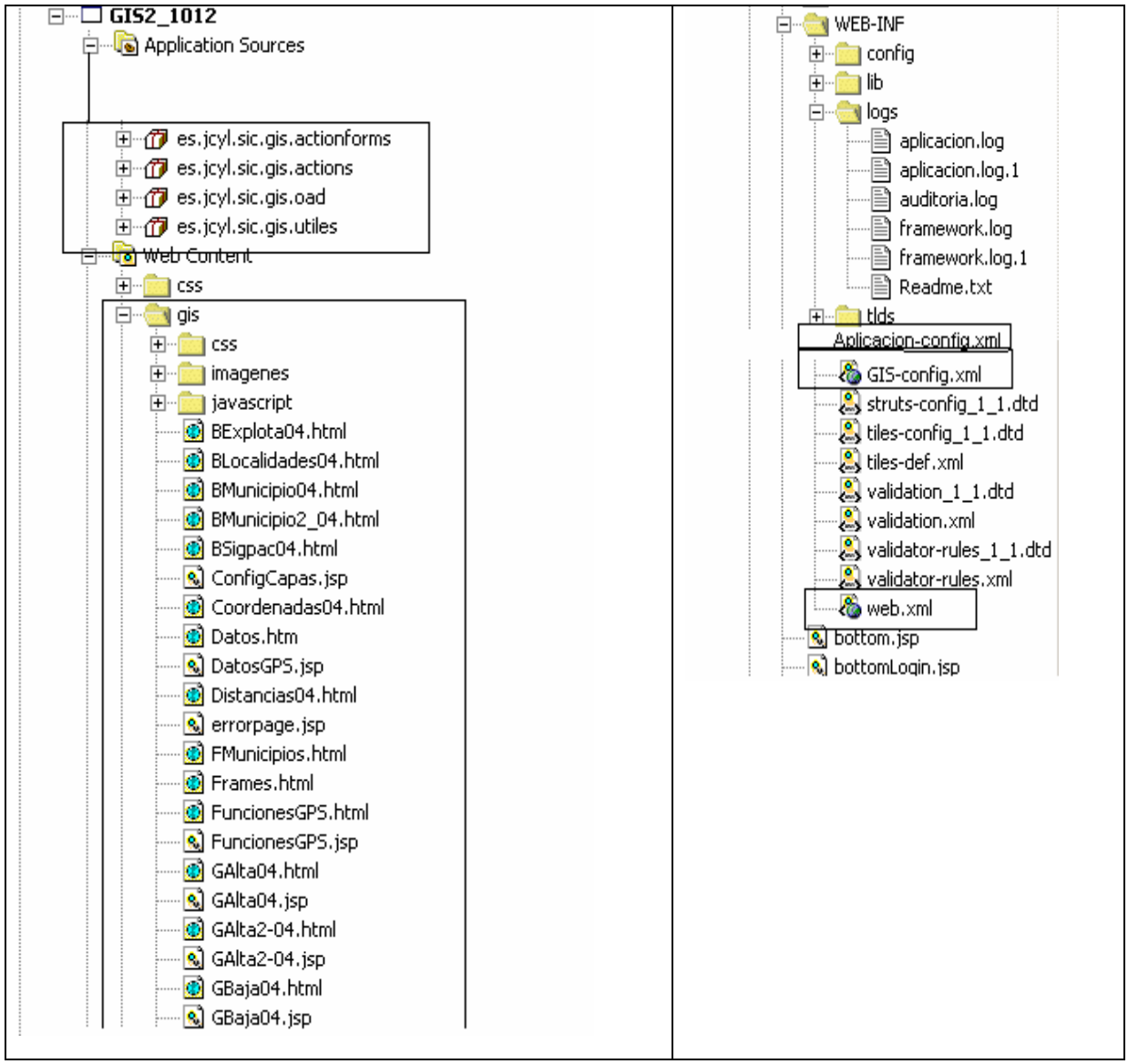


3.2.3 NOVEDAD: Extensión Modulo GIS

Como modulo opcional, puede añadir al esqueleto básico de aplicación la **extensión GIS.**

Esta extensión consiste en un amplio conjunto de acciones y elementos de programación debidamente configurados que le ayudarán a la hora de construir un aplicativo GIS para la Junta de Castilla y León.

Los detalles de este modulo y su uso puede consultarse en el documento **APPBASE_ConfigGISX_X.pdf**



El detalle

4 PERSONALIZAR LA APLICACIÓN BASE

Una vez generada nuestra aplicación base es necesario revisar y adaptarla a nuestras necesidades. Para ello se deben configurar los siguientes apartados:

4.1 Sistema de Logs. Cambios en WEB-INF\config\log.xml

Debe actualizarse el nombre y las rutas de los ficheros de logs que se desee crear en la aplicación. Por defecto nos los dejará en WEB-INF\logs. Para más información leer ESCRITURA EN FICHEROS DE LOGS.

4.2 Propiedades de la aplicación. Cambios en WEB-INF\config\ app-config.properties

En este archivo se deben modificar los valores –sólo los valores, no los nombres- de las propiedades que vienen incluidas por defecto en el mismo.

4.2.1 Propiedades generales de la aplicación

Propiedad	Descripción	Ejemplo de Valor
SISTEMA	Debe especificarse la clave de 4 Caracteres que identifica la aplicación y que es la utilizada en el sistema de seguridad.	GECE
TITULO	Etiqueta de Título la aplicación que aparece en la pantalla de login	
ClaseConfiguracion	Clase que inicializa la configuración propia de la aplicación. Aparece rellena por defecto	
TIEMPO_REFRESCO_ LOGS_ SEGUNDOS	Tiempo en segundos para volver a refrescar el fichero de logs de la aplicación (log.xml)	**60**

4.2.2 Propiedades relativas al sistema de seguridad

Propiedad	Descripción	Ejemplo de Valor
SEGU_HOST	Maquina (Dirección IP, o DNS) donde reside el servidor OC4J que contiene los componentes de Seguridad (Acceso REMOTO)	**localhost**
SEGU_PORT	Puerto para acceder al servidor OC4J como administrador (Acceso REMOTO)	**3001**
SEGU_USER	Identificación ante el servidor OC4J (Acceso REMOTO)	**admin**
SEGU_PASSWORD	Palabra clave (Acceso REMOTO)	**Xxxx**
SEGU_APP_SEGURIDAD	Nombre de la aplicación en el servidor donde ha sido desplegado el componente	**SistSegu**

SEGU_POOL	Repositorio de Datos de Segu por defecto	**pool-name**
SEGU_REFRESCO	Indica si al crear una nueva Sesion hay que refrescar el HastTable de funciones accesibles. Valores posibles: true=Entrada rapida sin refrescar las funciones fase = Actualiza cada entrada en Sesion	**false**

4.2.3 Otras propiedades

Propiedad	Descripción	Ejemplo de Valor
CON_AUDITORIA	Activar/Destactivar la AUDITORIA DE ORACLE 9i. Valores posibles SI/NO	**NO**
NOMBRE_DATASOURCE	Nombre del pool DataSource utilizado	**jdbc/DEMOPoolDS**
TIPO_LOGIN_DEFECTO	Tipo de autenticación que va a utilizar la aplicación. Valores posibles: publico, password, certificado	**password**
USUARIO_PUBLICO	Cuenta de usuario usada para autenticarse en el caso de acceso público	**anonimo**
PASSWORD_PUBLICO	Contraseña usada para autenticarse en el caso de acceso público	**anonimo**
TRIPLEDES	Clave del algoritmo TripleDes para Autenticación por certificado	
URL_AUTENTIFICACION_CERTIFICADO	URL del servidor de autentificación para validación con certificado	

4.2.4 Añadir propiedades a nuestra aplicación

Además pueden añadirse cuantos parámetros se desee. Posteriormente se podrá acceder a los mismos a través del método:

```
String valor = JCYLConfiguracion.getValor("<NOMBRE_PROPIEDAD");
```

4.3 Descriptor de controlador Struts. Archivo WEB-INF\struts-config.xml

En este archivo se deben definir los elementos form-bean y action a medida que se vaya construyendo la aplicación.

Si se desea introducir el control de validación de Struts se deben tener en cuenta las indicaciones del documento APPBASE_NotasNuevaVersionX_X.doc (X_X versión actual de este documento) relativas a este tema.

4.4 Otras consideraciones

- No se puede hacer una llamada directa a una página *.jsp.

- Para las **aplicaciones de acceso publico (NOVEDAD)**, solamente se requiere especificar las nuevas propiedades **TIPO_LOGIN_DEFECTO, USUARIO_PUBLICO y PASSWORD_PUBLICO.**
- En la clase que realice el **login** (la que sustituya a `es.jcyl.consejeria.NombreAplicacion.actions.NombreAplicacionLoginAction`) no debe alterarse el código que se presenta; sin embargo puede añadirse el código que se estime necesario.
- Se incorpora en código fuente la clase **JCYLConfiguracion.java** dentro del paquete es.jcyl.framework. Esta clase que se distribuía en versiones anteriores dentro de **jcylfwx_x.jar** se ofrece para consultar los pasos seguidos en arranque básico de una aplicación y el uso de las capacidades del componente ConfigurationRepository. Se recomienda no cambiar.

5 CARACTERÍSTICAS DE LA APLICACIÓN GENERADA

Una vez ejecutada la aplicación y hechos todos los cambios indicados, podemos entrar en JDeveloper y abrir el workspace generado. Este se encuentra dentro del directorio de la aplicación\fuentes.

Existen un fichero .jws (workspace) asociado a las distintas versiones de JDeveloper (9.0.3 y 9.0.5 o 10g). Es aconsejable renombrar el fichero .jws y .jpr eliminando la etiqueta de la versión del JDeveloper desde dentro de la herramienta de desarrollo.

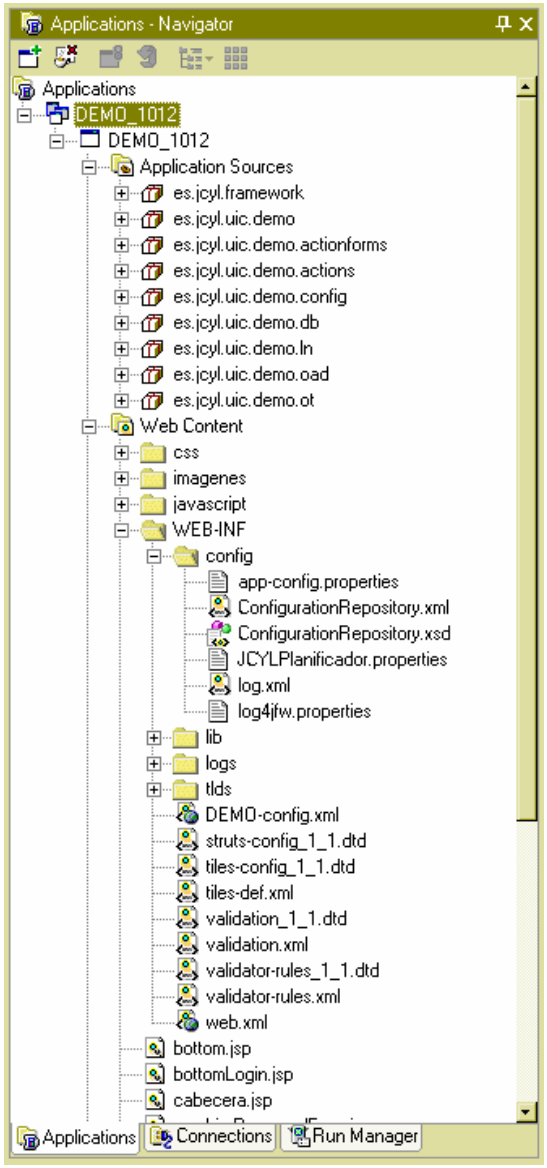


Figura 4: Proyecto Generado

La aplicación está lista para ejecutar.

Si ejecutamos entraremos en la pantalla de login y después de introducir un usuario/password de la aplicación nos mostrará la pantalla de menús (si estos ya están definidos en la B.D.).

Un ejemplo de ejecución es el siguiente:

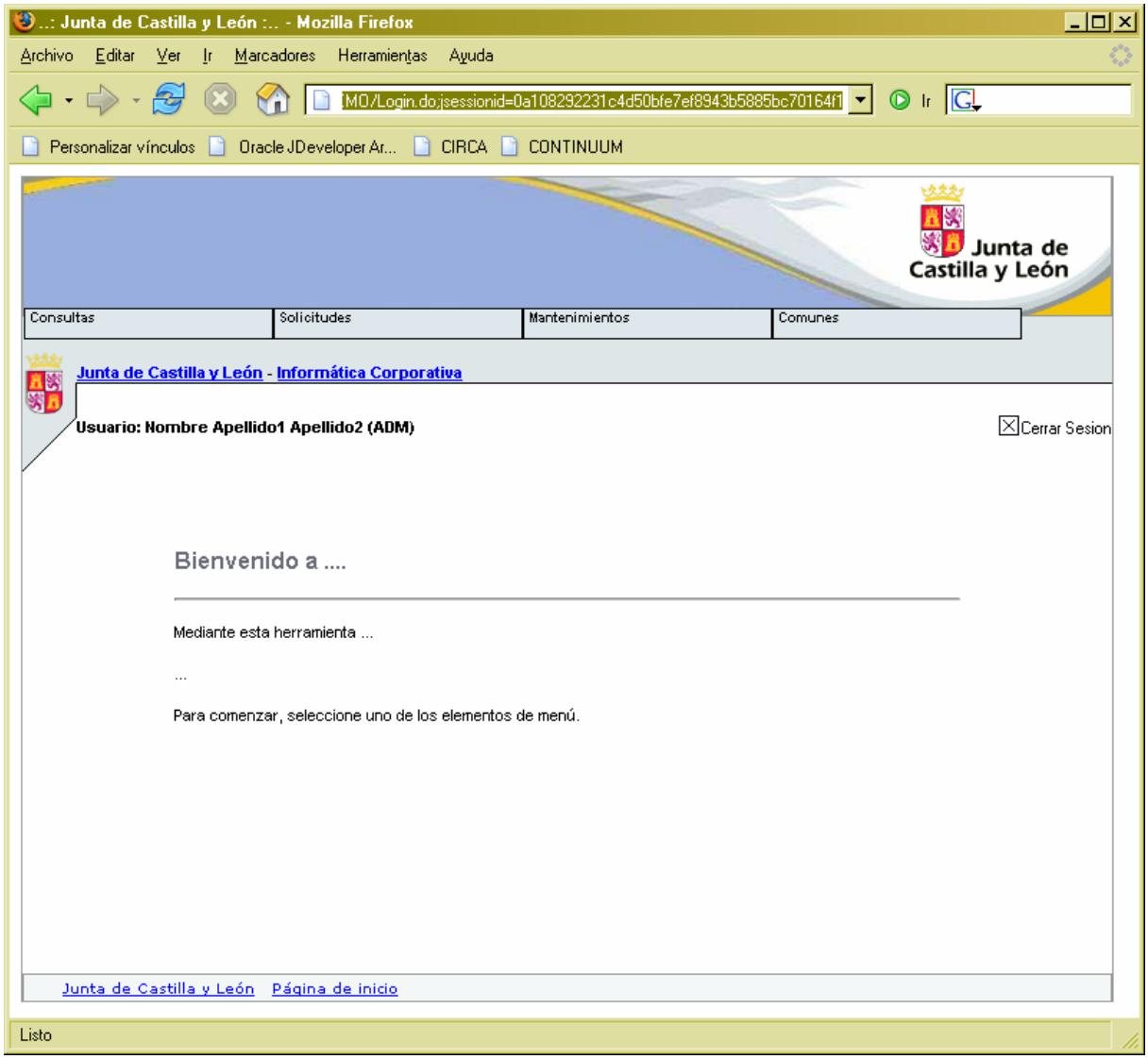


Figura 5: Aplicación Generada

6 DESCRIPCIÓN DE LOS COMPONENTES PRINCIPALES DISPONIBLES

6.1 Personalización en la secuencia de arranque

La versión actual del proyecto ofrece la clase es.jcyl.framework.JCYLConfiguracion.

Esta clase se estaba distribuyendo empaquetada en versiones anteriores al jcylfw-1.3.jar. La secuencia de pasos de arranque, **la cual se recomienda encarecidamente no alterar,** está implementado en el método **public static void inicializar(Servlet servlet)**.

La intención de esta mejora es hacer la secuencia de arranque mas transparente a los desarrolladores y mostrar el caso práctico de uso del nuevo componente **GestorConfiguracion.**

Para añadir operaciones especificas de inicializacion y finalización globales de la aplicación se recomienda seguir usando la clase XXXXConfigApp

6.1.1 Ejecución de código al inicio o fin de la aplicación. Clase XXXConfigApp

La aplicación base generada mediante esta herramienta nos define un entorno en el cual podemos definir que programa o código queremos que se ejecute tanto iniciar la aplicación como justo antes de la finalización del mismo.

Para ese propósito se ha definidos los métodos **inicializar** y **finalizar** de la clase *es.jcyl.consejeria.nombreaplicacion.config.NOMBREAPLICACIONConfigApp.java.*

El siguiente ejemplo ilustra como incluir código a ejecutar al inicio de la aplicación. Este mismo ejemplo lo detallaremos dentro del ANEXO II.

```
public void inicializar(ServletContext application) throws Exception {
  logger = JCYLConfiguracion.getLogger(JCYLConfiguracion.LOG_APLICACION);
  logger.debug(getClass().getName()+"  inicializar() ");

  /* Cargar lista de Provincias y almacenarlas en el contexto de la aplicación */
  DEMOConfigLN ln = new DEMOConfigLN();
  java.util.ArrayList  provincias_cyl = ln.RecuperarListaProvincias();
  application.setAttribute("PROVINCIAS_CYL", provincias_cyl);
  logger.debug("Guardardo en Aplicacion Provincia :" +provincias_cyl.toString())
  provincias_cyl = null;
/* Cargar lista de Consejerias y almacenarlas en el contexto de la aplicación */
  java.util.ArrayList  consejerias = ln.RecuperarListaConsejerias();
```

```
  application.setAttribute("CONSEJERIAS", consejerias);
  consejerias = null;
  ln = null;
}
```

6.2 Planificador de tareas

Se ha incluido en el framework base un reducido gestor de tareas. Este componente se ha diseñado tratando de reducir el trabajo del programador a la hora de generar y planificar tareas.

Este componente se ha diseñado tratando de reducir el trabajo del programador a la hora de generar las nuevas tareas. En este apartado se va a comentar la forma de crear una nueva tarea para que pueda ser utilizada por el planificador.

Toda tarea que se cree debe extender de la clase `JCYLTarea` e implementar los dos métodos abstractos de esta clase (`ejecutar()` y `setParametros()`) para realizar las acciones para las que se cree la tarea

- Método `ejecutar()`
  En este método se lleva a cabo todo el trabajo de la tarea. Se debe tener en cuenta que si la tarea es interrumpible debe ser el programador el que establezca puntos de control en los que se compruebe si se ha solicitado la cancelación de la tarea, en cuyo caso se anotará en el log y no se continuará con la ejecución del método. Si la tarea no es interrumpible, aunque se definan estos mismos puntos de control nunca se activarán puesto que los métodos de la clase padre consideran esta posibilidad.

- Metodo `setParametros()`
  En este método es donde se debe incluir el código necesario por la tarea para hacer uso de los parámetros definidos en el fichero de propiedades. Si no se necesita utilizar parámetros este método puede estar vacío, pero siempre hay que implementarlo.

Una vez creada una nueva tarea, y por tanto implementados los métodos anteriores, se debe incluir la información de la tarea en el fichero de propiedades del planificador para que éste la pueda utilizar.

Este está ubicado en **WEB-INF\config\JCYLPlanificador.properties** y en el se definen los siguiente aspectos:

- Activar / Desactivar el planificador
- Definir la lista de tareas (clases java) con sus aspectos de ejecución:

- Intervalo de tiempo entre ejecuciones (unidades de tiempo = minutos)
- Tarea activa o desactiva por defecto
- Tarea interrumpible o no
- Nº máximo de ejecuciones
- Relación de parámetros que necesite le tare

Si se ha decidido que la tarea puede ser interrumpida durante su ejecución se deben insertar puntos de control a lo largo del método ejecutar de forma que se comprueba si se ha solicitado la cancelación de la tarea y si es así se registra en el log y se sale del método permitiendo al planificador continuar con otras tareas. Para comprobar si se ha indicado cancelar se puede utilizar el método de la clase padre `comprobarSalir()`. A continuación se muestra el código ejemplo que se puede utilizar

```
if (comprobarSalir()) {
     JCYLPlanificadorTareas.log("Tarea cancelada");
     setCancelada(true); // !!!OJO!!! esto es fundamental
                                 // puesto que sino se puede bloquear el
                                 // planificador ya que está a la espera
                                 // que la tarea finalice correctamente
                                 // o bien que se haya indicado que la
                                 // cancelación se ha ejecutado
     return;
 }
```

Como se puede ver, también se debe indicar que la tarea ha finalizado para permitir al planificador continuar con otras ejecuciones puesto que sino quedará a la espera de la finalización de la tarea y esto no se producirá nunca.

Cuando se llama al método cancelar se detiene la ejecución del planificador hasta que la tarea haya finalizado o se haya llegado a un punto de control y se haya cancelado.

Ejemplo de una Tarea :

```
public class TareaEjemplo extends JCYLTarea {
    public TareaEjemplo() {
        super();
    }
    protected void ejecutar() {
        JCYLPlanificadorTareas.log("Tarea, " + 3 + ", " + this + ", timed " + 1);
        try {
            Thread.sleep(40000);
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }
        if (comprobarSalir()) {
            JCYLPlanificadorTareas.log("Tarea cancelada");
            setCancelada(true);
            return;
        }
        try {
            Thread.sleep(40000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        JCYLPlanificadorTareas.log("Tarea, " + 3 + ", " + this + ", finalizada");
    }
    public void setParametros(String[] parametros) {
    }
}
```

6.3 Sistema de Log Estándar (log4j)

La aplicación base incorpora un sistema de Log basado en el componente **log4j.**

La primera y una de las mayores ventajas de cualquier API de logging sobre el tradicional **System.out.println** es la capacidad de habilitar y deshabilitar ciertos logs, mientras otros no sufren ninguna alteración. Esto se realiza categorizando los mensajes de logs de acuerdo al criterio del programador.

Log4J tiene por defecto 5 niveles de prioridad para los mensajes de Log:

- **DEBUG**: Se utiliza para escribir mensajes de depuración, este log no debe estar activado cuando la aplicación se encuentre en producción.
- **INFO**: Se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.
- **WARN**: Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.
- **ERROR**: Se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **FATAL**: Se utiliza para mensajes críticos del sistema, generalmente luego de guardar el mensaje el programa abortará.

Adicionalmente a estos niveles de log, existen 2 niveles extras que solo se utilizan en el archivo de configuración, estos son:

- **ALL**: este es el nivel más bajo posible, habilita todos los logs.
- **OFF**: este es el nivel más alto posible, deshabilita todos los logs.

La aplicación base usa tres ficheros para almacenar información de tres tipos: aplicación, auditoria y framework. Estos archivos se encontrarán en el mismo directorio que tengamos definido en el fichero *log.xml.* Si no lo hemos modificado estará en WEB-INF\logs.

- El **fichero aplicación** contiene todos los mensajes de log que nosotros hayamos definido dentro de nuestra aplicación.
- El **fichero auditoria** contiene las opciones accedidas por cada usuario.
- El **fichero framework** contiene información sobre el funcionamiento interno del framework.

6.3.1 Uso del sistema de Log en la aplicación base

En toda clase en la que deseemos usarlo debemos importar la clase:

```
import es.jcyl.nombreconsejeria.nombreaplicacion.config.NOMBREAPLICACIONConfigApp;
```

Cuando queramos invocarlo debemos hacerlo de la siguiente manera:

```
NOMBREAPLICACIONConfigApp.logger.error(mensaje);
```

Error , puede sustituirse por debug, info, etc..

También podemos añadir todos los appenders a mayores que queramos

Para mas información sobre como usar el componente **log4j,** consultar el documento log4j.doc.

6.4 Validaciones basadas en Struts.

A partir de la versión 1.2 se han implementado los cambios necesarios en la **Aplicación Base** para poder utilizar las posibilidades de validación avanzada de formularios mediante Struts. Éste sistema de validación (**Struts Validator**) ya estaba disponible en la versión de Struts utilizada en la aplicación base (Struts 1.1) por lo que el siguiente apartado se encargará únicamente de documentar a grandes rasgos como usarlo.

Para mas detalles de las capacidades de **Struts Validator** se recomienda consultar el documento de ***Struts in Action*** apartado *4.9 Configuring the Struts Validator* y en el *apartado 12 Validating user input*, se van describir los pasos a realizar para implantar dicha solución.

6.4.1 Pasos para aplicar la validación basada en Struts

Se van describir los pasos a realizar para implantar dicha solución para la validación de campos tanto en la parte cliente como en la parte servidora, ambas validaciones son independientes por lo que en los puntos se indicaran cuales pertenecen a la **validación cliente**[1] y cuales a la **validación de servidor**[2] y cuando no se indique nada es que pertenecen a las dos.

6.4.2 Actuaciones en los XML con las reglas de validación

Los pasos a seguir añadir una validación basada en reglas con Struts son los siguientes:

1. Añadir en /WEB-INF los ficheros **validator-rules.xml** y **validation.xml** facilitados con la distribución de la **aplicación base 1.2 o posteriores**

   Nota: Para obtener estos ficheros se debe generar una aplicación base vacía en un directorio temporal con el mismo nombre de aplicación que la que se está desarrollando y obtener los ficheros indicados.

2. Del fichero **validation.xml** eliminar, si no se desea utilizar, la información relacionada con las validaciones de los formularios LoginForm (**<form name= "LoginForm"> … </form>**) y CambioPasswordForm (**<form name= "CambioPasswordForm"> … </form>**). Se pueden consultar para ver como está implementado en la aplicación base.

3. Insertar en **/WEB-INF/*nombreAplicación*-config.xml** inmediatamente antes del tag **</struts-config>** se debe introducir el tag siguiente:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
<set-property
   property="pathnames"
   value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

4. Añadir en el validation.xml una sección <formset> por locale (idioma) como formularios se quieran validar introduciendo entre los tag <form-validation> <form-validation> un código equivalentes al siguiente:

```
    <formset>
        <form name="nombreFormulario">
```

```
            <field     property="nombrePropiedad"
                              depends="tiposDependencias">
            <arg0 key="${}"/>
            </field>
        </form>
    </formset>
```

Cada formulario tendrá su sección **<form name=””> … </form>** por cada formulario que se desee validar.

También tendrá una sección y un **<field> … </field>** por cada campo del formulario que se desee validar. Teniendo en cuenta que los ***tiposDependecias*** pueden ser varios y deben ir separados por comas.

Los tipos de dependencias son las siguientes:

required	minlength	maxlength
invalid	byte	short
integer	long	float
double	date	range
creditcard	email	mask

Para más información de cómo configurar este fichero fijarse en los ejemplos dados en el validation.xml de la aplicación base o en el documento mencionado anteriormente ***Struts in Action***.

En resumen, este fichero es en el que se configuraran todos los campos de todos los formularios donde se quiera hacer validaciones y las validaciones que se deseen asociar a cada campo.

6.4.3 Actuaciones en los ActionsForms y AplicacionResources.properties

En los ActionsForms hay que indicarles que usen estas reglas de validación. Para eso es necesarios revisar:

1. Hacer que los Form que se deseen validar en la parte servidora hereden de **org.apache.struts.validator.ValidatorForm** en vez de **org.apache.struts.action.ActionForm** [2] (Para validación en servidor)
2. Comprobar que la clase compila revisando las entradas **import**

```
    import org.apache.struts.validator.ValidatorForm;
```

3. Comprobar que al menos existe el siguiente código en el método **validate(ActionMapping mapping, HttpServletRequest request)** del Form [2] (Para validación en servidor).

```
    ActionErrors errores=super.validate(mapping,request);
    if (errores==null)
        errores = new ActionErrors();
    //INICIO: Aquí se introducir el código con los controles
```

```
    //añadidos que se deseen realizar
    //FIN.
    return errores;
```

4. Comprobar que en el fichero **AplicationResources.properties** los mensajes asociados a las validaciones introducidos en el **validation.xml** o si se quieren usar los de por defecto introducir la lista siguiente:

```
# Struts Validator Basic Error Messages

errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {1} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be a byte.
errors.short={0} must be a short.
errors.integer={0} must be an integer.
errors.long={0} must be a long.
errors.float={0} must be a float.
errors.double={0} must be a double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is not a valid e-mail address.
```

6.4.4 Actuaciones en la JSP del formulario y validaciones en cliente

Para la **Validación en cliente**[1] existen varias posibilidades dependiendo de cómo se construya la página:

Si existe tag **<html:submit>** se debe:

- Añadir en el tag **<html:form>** el atributo **onsubmit= "validate*NombreFormulario*(this)"**
- Añadir justo después del tag **</html:form>** el tag:

```
<html:javascript formName="NombreFormulario" />
```

Si **no existe tag <html:submit>** se debe:

- Revisar todos los métodos javaScript donde exista la sentencia document.forms[0].submit(); o alguna otra equivalente y añadir a la lógica que tuviese si se cree conveniente hacer la validación en cliente en vez de la sentencia anterior lo siguiente:

```
if(validateNombreFormulario(document.forms[0])){
   document.forms[0].submit();
 }
```

- Además como en el caso anterior justo después del tag **</html:form>** el tag siguiente:

```
<html:javascript formName="NombreFormulario" />
```

Y Recuerda que:

- Debes comprobar en el fichero **/WEB-INF/*nombreAplicación*-config.xml** que el formulario con el que estas trabajando se valide con Struts e indicar la página JSP en la cual retornar en caso de no superar la validación. Esto es, tener los valores *validate=true* e *input="pagina.jsp".* Ejemplo :

```
<action path="/DEMOSolVacacionesAl"
                type="es.jcyl.uic.demo.actions.DEMOSolVacaciones"
                input="DEMOSolVacaciones.jsp"
                name="DEMOFormularioForm"
                scope="request"
                validate="true">
                <forward name="exito" path="/DEMOSolVacaciones.do"/>
</action>
```

- Comprobar que en la página JSP del formulario están las tags donde presentar los mensajes de error definidos en la validación, como por ejemplo:

```
<!-- Propiedad "DNombre" -->
    <tr><td class="textoTituloGris" align="left">
        DNombre
    </td><td align="left">
        <html:text property="DNombre"/> <html:errors property="DNombre" />
    </td></tr>
```

6.5 Gestión de transacciones en aplicación Base.

El objetivo principal de ésta nueva propuesta es mejorar la Gestión de transacciones entre objetos OAD's. Con ella, se pretende no tener que ir pasando la conexión y el usuario por toda las clase de lógica de negocio y OAD's que involucren a una transacción.

La filosofía propuesta para conseguir esto es **asociar tanto la conexión como el usuario al thread que se esta utilizando que en una transacción siempre es el mismo**.

En su implementación se han modificado clases generadas en la aplicación base como nuevas versiones y nuevas clases de las librerías internas al framework: (**jcylfw-1.1.jar**, **jcylutiles-1-1.jar**, **jcyldb-1.0.jar, o posteriores**).

El detalle de esas actuaciones es :

- Relacionado con la asociación del usuario al thread.
  - **es.jcyl.framework.JCYLUsuario** (Encargada de asociar al thread el usuario)
  - **es.jcyl.framework.JCYLRequestProcessor** (Encargada de asociar el usuario que se encuentra en la sesión al thread actual)
  - **NombreAplicacionLoginAction** (Encarga de asociar el usuario que se encuentra introduce en la sesión por primera vez al thread actual)
  - **NombreAplicacionConexionDB** (Encarga de realizar la auditoria si fuese necesario para todo tipo de conexión)

- Relacionado con la asociación de la conexión al thread.
  - **NombreAplicacionPruebaAccesoBDOAD** (se ha convertido a patrón singelton y se controlan las transacciones basándose en el clase JCYLGestionTransacciones)
  - **NombreAplicacionPruebaAccesoBDLN** (no se ha convertido a patron singelton pero es recomendable hacerlo por eficiencia y se controlan las transacciones basándose en el clase JCYLGestionTransacciones)
  - **NombreAplicacionConexionDB** (se controlan las conexiones basándose en el clase JCYLGestionTransacciones)

Para ello se han creado las siguientes clases del framework

- Relacionado con la asociación del usuario al thread.
  - **es.jcyl.framework.utiles.JCYLGestionUsuario** (es la que se utilizara para obtener el usuario asociado al thread donde nos encontramos)

- Relacionado con la asociación de la conexión al thread.
  - **es.jcyl.framework.db.JCYLDataBase** (es la que se utilizara para basada en el patrón Proxy obtener la conexión y la asocia al thread en el que se encuentre a través de la clase JCYLGestionTransacciones)

- **es.jcyl.framework.db.JCYLGestionTransacciones** (es la que se utilizara para controlar las transacciones asociadas al thread en el que nos encontramos)

Además se ha modificado el generador de OAD's para que incluya las modificaciones oportunas, para implantar lo explicado como se indica en el apartado siguiente.

La filosofía de de esta solución se basa en asociar objetos al thread que se este ejecutando (usuario para auditorias, conexión para transacciones) y con ello no hace falta pasarlos a las clases de lógica de negocio ni a los OAD's por tanto si dentro de una misma clase de lógica de negocio se quiere realizar una transacción al principio del método de esta lógica de negocio se abre la transacción (**open**) y al final del método se realiza el **commit** si ha ido bien la transacción o **rollback** si ha fallado la transacción y se cierra la transacción(**close**), mientras que en los OAD's se obtiene la conexión que ha sido abierta en la lógica de negocio y se utiliza.

Si la transacción es tan compleja que implica a varias lógicas de negocio existen dos posibles soluciones basándonos en esta filosofía:

1. Realizar una clase de lógica de negocio "genérica" que se encarga de abrir y cerrar transacción y en medio de estas llamadas invocar a las distintas lógicas de negocio involucradas.
2. Que sea el action el que se encarga de abrir y cerrar transacción y en medio de estas llamadas invocar a las distintas lógicas de negocio involucradas en la transacción.

6.6 NOVEDAD. Gestión dinámica de configuración de la aplicación

6.6.1 Introducción

Todas las aplicaciones tienen la necesidad de almacenar parámetros operativos de forma externa al código, con el objetivo de facilitar su despliegue en entornos de producción y permitir la adecuada integración con elementos externos (servidores de correo, de ficheros, otras aplicaciones, etc.)

El framework de desarrollo para aplicaciones J2EE de la Junta de Castilla y León gestiona la configuración de una forma centralizada mediante el componente GestorConfiguración descrito en este documento. En esta versión se ofrecen **ya configurado el componente** de forma que tenga un comportamiento y ubicación equivalente a las que tenía en versiones anteriores.

Las principales características que relacionan este componente son:

- Permite a las aplicaciones almacenar su configuración tanto en ficheros de propiedades (.properties) como en ficheros en xml.
- Permite gestionar de forma transparente múltiples ficheros de configuración. Los parámetros pueden estar en uno o varios ficheros, que pueden agruparse por contextos o tratarse de forma centralizada.
- La aplicación tiene acceso a los cambios realizados en los ficheros de configuración sin necesidad de reiniciar la ejecución (recarga en caliente).
- La memoria ocupada por las colecciones de parámetros de una aplicación se optimiza siempre al máximo, gracias a un mecanismo dinámico de carga y desalojo de contextos de configuración.

El acceso a los ficheros de configuración y la gestión de los mismos se realiza gracias a las librerías proporcionadas por el proyecto **commons-configuration** del proyecto Jakarta.

http://jakarta.apache.org/commons/configuration/

6.6.2 Guía Rápida de uso

En este apartado se presenta un ejemplo sencillo de utilización del componente de configuración.

La referencia detallada puede consultarse en el manual específico del componente.

6.6.2.1 Elementos para configuración de la aplicación

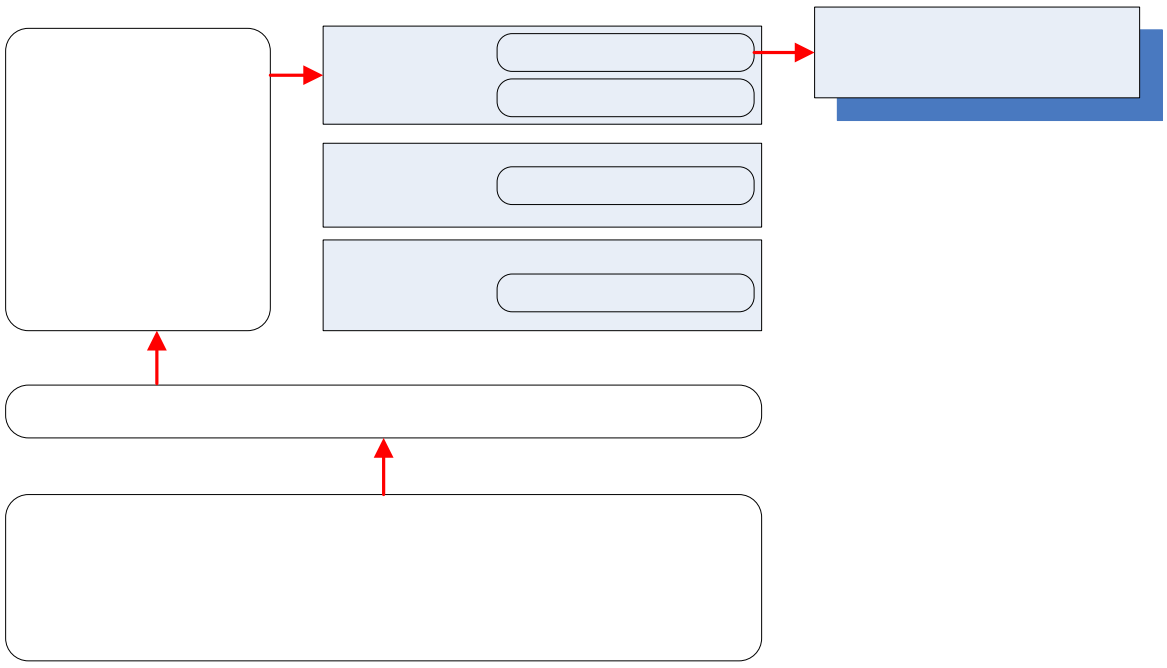
Para que una aplicación haga uso del sistema de configuración, debe incorporar un fichero denominado ***repositorio de configuraciones***, que contiene la referencia a todos los ficheros de configuración de la aplicación.

Para desacoplar la aplicación de la forma de ubicar los ficheros, este repositorio define el concepto de **contexto** como una agrupación de ficheros de configuración. Cuando la

aplicación desee leer un parámetro de configuración lo hará especificando el contexto en que se encuentra.

La configuración de la aplicación como tal se encuentra repartida en uno o varios **ficheros**, siempre asociados a un contexto. Este componente soporta los tradicionales ficheros .properties y ficheros XML

Esquemáticamente puede verse el conjunto en este diagrama:



Un ejemplo sencillo del fichero que define el repositorio se representa a continuación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configurations version="1.0" xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance' xsi:noNamespaceSchemaLocation='/xsd/ConfigurationRepository.xsd'>

   <!-- Configuraciones globales -->
   <globals>
       <!-- Ruta base para todos los ficheros de configuración -->
       <initialPath value="${web-inf.directory}/config/"/>

       <!-- Propiedades globales -->
       <properties>
           <property name="minimumCheckDelay" value="10000"/>
           <property name="contextTimeOut" value="10000"/>
       </properties>
   </globals>

   <!-- Lista de contextos de la aplicación -->
   <contexts>
       <!-- Configuración de la aplicación -->
       <context id="default" name="Configuración de la aplicación">

           <configurationFile
                    id="application"
                    name="Parámetros generales de la aplicación"
                    location="/pruebas/config/prueba.properties"/>

       </context>

       <!-- Configuración del módulo de PDF -->
       <context id="pdf" name="Componente de generación de documentos PDF">

           <configurationFile
                    id="application"
                    name="Parámetros del componente"
                    location="./pdf.xml"/>
       </context>

   </contexts>
</configurations>
```

Este repositorio contiene los siguientes elementos:

- Una lista de propiedades globales que permiten la parametrización del propio gestor de configuraciones.

```
<!-- Configuraciones globales -->
<globals>
   <!-- Ruta base para todos los ficheros de configuración -->
   <initialPath value="${web-inf.directory}/config/"/>

   <!-- Propiedades globales -->
   <properties>
       <property name="minimumCheckDelay" value="10000"/>
       <property name="contextTimeOut" value="10000"/>
   </properties>
</globals>
```

- Un contexto **default** que para la configuración básica de la arquitectura de la aplicación. Contiene un único fichero de configuración basado en un fichero de propiedades estándar, denominado **prueba.properties**.

```
<!-- Configuración de la aplicación -->
<context id="default" name="Configuración de la aplicación">

    <configurationFile
         id="application"
         name="Parámetros generales de la aplicación"
         location="/pruebas/config/prueba.properties"/>
</context>
```

- Un contexto **pdf** para la configuración de un componente para generación de ficheros en formato pdf. Contiene un único fichero de configuración basado en xml denominado **pdf.xml**.

```
<!-- Configuración del módulo de PDF -->
<context id="pdf" name="Componente de generación de documentos PDF">

    <configurationFile
         id="application"
         name="Parámetros del componente"
         location="./pdf.xml"/>

</context>
```

6.6.2.2 Acceso a la configuración desde la aplicación

Para permitir que la aplicación localice el fichero con el repositorio de configuraciones debe ser inicializada indicando la ruta del fichero.

Este proceso debe realizarse una única vez durante la ejecución de la aplicación. Obsérvese que el uso del componente GestorConfiguración se realiza mediante métodos estáticos, por lo que no es necesario almacenar ninguna referencia al mismo.

6.6.2.3 Inicialización para aplicaciones web

En el caso de aplicaciones web, basta con añadir una nueva entrada en el fichero de despliegue web.xml de su aplicación, indicando la ruta donde se encuentra:

```
  <env-entry>
     <env-entry-name>configurationRepository</env-entry-name>
     <env-entry-value>applications/demo/demo/WEB-
INF/config/ConfigurationRepository.xml</env-entry-value>
     <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

Desde la aplicación debe obtener este valor y emplearlo para inicializar el componente GestorConfiguracion.

```
// Obtiene la ruta al configuration repository
String ruta = getEnvEntry("configurationRepository");

// Inicializa GestorConfiguracion a partir de la ruta donde se encuentra
GestorConfiguracion.initialize( ruta );
```

7 ANEXO I

7.1 EJEMPLO 1. Pasos para utilizar PreparedStatement en los OAD's

Por temas de seguridad, se recomienda sustituir la llamada al método createStatement por prepareStatement.

A continuación se muestra cómo quedarían el OAD's de ejemplo que se encuentra al desplegar la aplicación base, para más información se recomienda tener en cuenta las indicaciones del documento anteriormente citado.

```
package es.jcyl.uic.demo.oad;

import java.sql.*;
import java.util.*;
import java.util.Date;
import es.jcyl.uic.demo.ot.*;
import es.jcyl.cf.seguridad.util.Usuario;
import org.apache.log4j.Logger;

import es.jcyl.uic.demo.config.DEMOConfigApp;
import es.jcyl.framework.db.*;

/**
 * Clase objeto de acceso a datos para la tabla Provincias.
 * Hay que verificar en futuras versiones si nos interesa que las
 * funciones sean estaticas
 */
public class ProvinciasOAD {

     public static Logger logger = DEMOConfigApp.logger;
     private static ProvinciasOAD instance;
     public static ProvinciasOAD getInstance() {
            if (instance==null)
                   instance= new ProvinciasOAD();
            return instance;
     }

     private ProvinciasOAD()  {
      super();
}
```

```
/**
 * Funcion de alta para la tabla <I>PROVINCIAS</I>.
 * @param datos OT con los datos a insertar
 * @returns int El numero de filas insertadas (1 en caso correcto)
 * @throws SQLException En caso de error con la Base de Datos
 */
public int altaProvincias(ProvinciasOT datos) throws SQLException {
    PreparedStatement st = null;
    Connection con=JCYLGestionTransacciones.getConnection();

    int filas = 0;

           try {
              String sql = "INSERT " +
                 "INTO PROVINCIAS (A_ABREV_PR, A_ABREV_CA, D_PROV, C_PROV_ID) " +
                 "VALUES (?, ?, ?, ?)";

                  st = con.prepareStatement(sql);
                  st.setString(1, datos.getAAbrevPr());
                  st.setString(2, datos.getAAbrevCa());
                  st.setString(3, datos.getDProv());
                  st.setString(4, datos.getCProvId());


           filas  = st.executeUpdate();

           } catch (SQLException ex) {
                  throw ex;
           } finally {
                  JCYLGestionTransacciones.close(con.getAutoCommit());
           } // try-catch

           return filas;

    }


/**
 * Funcion de listado para la tabla <I>PROVINCIAS</I>.
 * @param inicio Registro a partir del cual devolver. <I>1</I> para primero.
 * @param cuantos Numero de registros a devolver. <I>-1</I> para todos.
 * @returns Un ArrayList con OT con el contenido de la tabla.
 * @throws SQLException En caso de error con la Base de Datos
 */
public ArrayList listadoProvincias(int inicio, int cuantos) throws SQLException {
    PreparedStatement st = null;
    ResultSet rs = null;
    Connection con=JCYLGestionTransacciones.getConnection();

    ArrayList listado = null;

    try {
           String sql = "SELECT A_ABREV_PR, " +
                                "A_ABREV_CA, " +
                                "D_PROV, " +
                                "C_PROV_ID " +
                        "FROM PROVINCIAS";

           st = con.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
           ResultSet.CONCUR_UPDATABLE);
           rs  = st.executeQuery();

           // Posicionar el cursor antes de inicio
                  if (inicio >1)
```

```
                        rs.absolute(inicio-1);

                listado = new ArrayList();
                int i = 0;
                while (rs.next()) {
                        ProvinciasOT datos = new ProvinciasOT();
                        datos.setAAbrevPr(rs.getString("A_ABREV_PR"));
                        datos.setAAbrevCa(rs.getString("A_ABREV_CA"));
                        datos.setDProv(rs.getString("D_PROV"));
                        datos.setCProvId(rs.getString("C_PROV_ID"));

                        listado.add(datos);

                        if (++i == cuantos)
                                break;

                } // while

        } catch (SQLException ex) {
                throw ex;
        } finally {
                JCYLGestionTransacciones.close(con.getAutoCommit());
        } // try-catch

        return listado;

    }

}
```